Project 4

Zhicheng Zhang - G45149856

1. Introduction

CUDA Matrix Multiplication - dot product of two square matrix.

2. Environment

- Intel(R) Core(TM) i7-9700K @ 3.60 GHz
- NVIDIA GeForce RTX 2080 Super
- Windows 10
- CUDA Toolkit 10.2
- Visual Studio 2019

3. Implementation

Data Structure of Matrix

Class Matrix in file matrix.h defines the data structure of the matrix:

- data is pointed to a dynamic allocated memory space which can could store total_data / size * size integers.
- There are two constructor:
 - Matrix(int size) is used to allocate memory space and generate a size * size matrix by filling random integers in [0, 100).
 - Matrix(int size, int fill) is used to allocate memory space generate a size *
 size matrix by filling fill.

```
#pragma once

class Matrix
{
  public:
    int *data;
    int size;
    int total_size;
    Matrix(int size);
    Matrix(int size, int fill);
    ~Matrix();
    void Show();
};
```

Dot Product By CPU

Function DotProductByCpu in file cpu.cpp uses 3-level for-loop to calculate dot product on CPU.

```
#include <windows.h>
#include "matrix.h"
#include "timer.h"
void DotProductByCpu(Matrix *input_1, Matrix *input_2, Matrix *output)
  int size = output->size;
  int total_size = output->total_size;
  int row;
  int column;
  int result;
  for (int i = 0; i < total_size; i++)</pre>
    row = i / size;
    column = i % size;
    result = 0;
    int temp = input_1->data[0];
    for (int j = 0; j < size; j++)
      int x = input_1->data[row * size + j];
      int y = input_2->data[j * size + column];
```

```
result += input_1->data[row * size + j] * input_2->data[j * size +
column];
    }
    output->data[i] = result;
}
```

Dot Product By GPU

Function DotProductByGpu in file gpu.cu indicate the way to calculate dot product By GPU.

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include "calculate.h"
#include "cuda_helper.h"
#include "matrix.h"
__global__ void KernelDotProduct(int *in_1, int *in_2, int size, int *out)
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  int total_size = size * size;
  if (index >= total_size)
   return;
  }
  int row = index / size;
  int column = index % size;
  int result = 0;
  for (int i = 0; i < size; i++)
    result += in_1[row * size + i] * in_2[i * size + column];
  }
  out[index] = result;
}
void DotProductByGpu(Matrix *input_1, Matrix *input_2, Matrix *output)
  int size = output->size;
  int total_size = output->total_size;
  CudaErrorHandler(cudaSetDevice(0));
  // allocate
  int *in_1 = NULL;
  int *in_2 = NULL;
  int *out = NULL;
  CudaErrorHandler(cudaMalloc(&in_1, total_size * sizeof(int)));
  CudaErrorHandler(cudaMalloc(&in_2, total_size * sizeof(int)));
  CudaErrorHandler(cudaMalloc(&out, total_size * sizeof(int)));
  // host => device
  CudaErrorHandler(cudaMemcpy(in_1, input_1->data, total_size * sizeof(int),
cudaMemcpyKind::cudaMemcpyHostToDevice));
  CudaErrorHandler(cudaMemcpy(in_2, input_2->data, total_size * sizeof(int),
cudaMemcpyKind::cudaMemcpyHostToDevice));
  // Launch a kernel on the GPU with one thread for each element.
  KernelDotProduct<<<total_size / 1024 + 1, 1024>>>(in_1, in_2, size, out);
  // Check for any errors launching the kernel
  CudaErrorHandler(cudaGetLastError());
  // host <= device
```

```
CudaErrorHandler(cudaMemcpy(output->data, out, total_size * sizeof(int),
cudaMemcpyKind::cudaMemcpyDeviceToHost));
// synchronize
CudaErrorHandler(cudaDeviceSynchronize());
}
```

Main

Function main in file main.cpp does the following things:

- 1. List parameters of CUDA devices.
- 2. Calculate "16 x 16 matrix dot production, M = all one's, N=all two's" by using CPU and GPU. Show time and result after the calculation.
- 3. Calculate dot production of groups of two matrices whose shade are 1x1, 2x2, 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, 2048x2048, 4096x4096 and 8192x8192 by using CPU and GPU. Items of matrices are randomly generated. Show time and result (only for matrices whose shapes are not larger than 8x8) after the calculation.

Source Code

I have uploaded the source code to my GitHub account. Here is it:

https://github.com/zzc-tongji/gwu-csci-6461-computer-system-architecture/tree/master/project-4

4. Result

```
CUDA device:
ASCII string identifying device:
                                            GeForce RTX 2080 SUPER
Number of asynchronous engines:
Clock frequency in kilohertz:
                                            1815000
Major and minor compute capability:
                                            7.5
Maximum size of each dimension of a grid:
                                           2147483647 / 65535 / 65535
Maximum size of each dimension of a block:
                                            1024 / 1024 / 64
Maximum resident threads per multiprocessor:
                                           1024
Maximum number of threads per block:
                                            1024
Maximum pitch in bytes allowed by memory copies: 2147483647
Number of multiprocessors on device:
32-bit registers available per block:
                                            65536
Shared memory available per block in bytes:
                                           49152
Alignment requirement for textures:
                                            512
Constant memory available on device in bytes:
                                           65536
Global memory available on device in bytes:
                                           8589934592
Warp size in threads (per SM):
size = 16x16
[CPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 29
Time (second): 0.000003
[GPU]
clock Rate (MHz): 9.000
Clock Count (tick): 1369
Time (second): 0.000137
\lceil m \rceil
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[n]
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2
       2
       2
        2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[r_cpu]
[r_gpu]
_____
size = 1x1
[CPU]
Clock Rate (MHz): 9.000
```

```
Clock Count (tick): 0
Time (second): 0.000000
[GPU]
clock Rate (MHz): 9.000
Clock Count (tick): 1477
Time (second): 0.000148
[m]
     56
[n]
     56
[r_cpu]
  3136
[r_gpu]
  3136
size = 2x2
[CPU]
clock Rate (MHz): 9.000
Clock Count (tick): 0
Time (second): 0.000000
[GPU]
clock Rate (MHz): 9.000
Clock Count (tick): 4725
Time (second): 0.000472
[m]
     56 38
          3
     44
[n]
        38
     56
     44
[r_cpu]
  4808 2242
  2596 1681
[r_gpu]
  4808 2242
  2596 1681
size = 4x4
[CPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 2
Time (second): 0.000000
```

Time (se	cond):	0.00014	9						
[m]									
56	38		3						
88	61	92	69						
80	51	23	93						
73	44	43	21						
[n]									
56	38	44	3						
88	61	92	69						
80	51	23	93						
73	44	43	21						
[r_cpu]									
	6822	7101	6945						
22693	14793	14567	14478						
17597	11416	12740	7851						
12933	8575	9152	7695						
[r_gpu]									
10219	6822	7101	6945						
22693		14567							
17597			7851						
	8575	9152	7695						
12933 size = 8 [CPU]		9152	7695						
 size = 8	 x8								
size = 8 [CPU] Clock Ra Clock Co	x8 te (MHz): 9.00 ck): 5	0						
size = 8 [CPU]	x8 te (MHz): 9.00 ck): 5	0						
size = 8 [CPU] Clock Ra Clock Co Time (se	x8 te (MHz): 9.00 ck): 5	0						
size = 8 [CPU] Clock Ra Clock Co	x8 te (MHz unt (ti	e): 9.00 ck): 5 0.00000	0						
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra	ex8 te (MHz tunt (ti cond):	c): 9.00 ck): 5 0.00000	0						
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co	ex8 te (MHz ount (ti cond): te (MHz	c): 9.00 ck): 5 0.00000 c): 9.00 ck): 82	0 0 0						
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se	ex8 te (MHz ount (ti cond): te (MHz	c): 9.00 ck): 5 0.00000 c): 9.00 ck): 82	0 0 0						
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se	ex8 Ite (MHz Bunt (ti Bunt (ti Bunt (ti Bunt (ti	ck): 9.00 ck): 5 0.00000 ck): 9.00 ck): 82 0.00008	0 0 0 7 3	88	61	92	69		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se	ex8 te (MHz ount (ti cond): te (MHz	c): 9.00 ck): 5 0.00000 c): 9.00 ck): 82	0 0 0	88 73	61 44	92 43	69 21		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se	ex8 Ite (MHz Punt (ti Ite (MHz I	ck): 9.00 ck): 5 0.00000 ck): 9.00 ck): 82 0.00008	0 0 0 7 3						
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se [m] 56 80	ex8 te (MHz cunt (ti cond): te (MHz cunt (ti cond): 38 51	c): 9.00 ck): 5 0.00000 ck): 9.00 ck): 82 0.00008	0 0 0 7 3	73	44	43	21		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se [m] 56 80 79 46 91	ex8 Ite (MHz Funt (ti Ficond): Ite (MHz Funt (ti Ficond): 38 51 66 33 48	(i): 9.00 ck): 5 0.00000 ck): 82 0.00008 44 23 81 49 78	0 0 0 7 3 3 93 83 12 69	73 90 1 0	44 22 92 28	43 80 97 1	21 7 68 82		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se [m] 56 80 79 46 91 53	ex8 Lite (MHz L	2): 9.00 ck): 5 0.00000 ck): 9.00 ck): 82 0.00008 44 23 81 49 78 91	0 0 0 7 3 3 93 83 12 69 42	73 90 1 0 96	44 22 92 28 42	43 80 97 1 42	21 7 68 82 11		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se [m] 56 80 79 46 91 53 33	ax8 Ite (MHz Funt (ti Ficond): Ite (MHz Funt (ti Ficond): 38 51 66 33 48 94 4	2): 9.00 ck): 5 0.00000 ck): 82 0.00008 44 23 81 49 78 91 83	0 0 0 7 3 3 93 83 12 69 42 46	73 90 1 0 96 69	44 22 92 28 42 15	43 80 97 1 42 14	21 7 68 82 11 94		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se [m] 56 80 79 46 91 53	ex8 Lite (MHz L	2): 9.00 ck): 5 0.00000 ck): 9.00 ck): 82 0.00008 44 23 81 49 78 91	0 0 0 7 3 3 93 83 12 69 42	73 90 1 0 96	44 22 92 28 42	43 80 97 1 42	21 7 68 82 11		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se [m] 56 80 79 46 91 53 33 46	ax8 Ite (MHz Funt (ti Ficond): Ite (MHz Funt (ti Ficond): 38 51 66 33 48 94 4	2): 9.00 ck): 5 0.00000 ck): 82 0.00008 44 23 81 49 78 91 83	0 0 0 7 3 3 93 83 12 69 42 46	73 90 1 0 96 69	44 22 92 28 42 15	43 80 97 1 42 14	21 7 68 82 11 94		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se [m] 56 80 79 46 91 53 33	ax8 Ite (MHz Funt (ti Ficond): Ite (MHz Funt (ti Ficond): 38 51 66 33 48 94 4	2): 9.00 ck): 5 0.00000 ck): 82 0.00008 44 23 81 49 78 91 83	0 0 0 7 3 3 93 83 12 69 42 46	73 90 1 0 96 69	44 22 92 28 42 15	43 80 97 1 42 14	21 7 68 82 11 94		
size = 8 [CPU] Clock Ra Clock Co Time (se [GPU] Clock Ra Clock Co Time (se [m] 56 80 79 46 91 53 33 46 [n]	2x8 Ate (MHz Bunt (ti Becond): 38 51 66 33 48 94 4 64	2): 9.00 ck): 5 0.00000 ck): 9.00 ck): 82 0.00008 44 23 81 49 78 91 83 34	0 0 0 7 3 3 93 83 12 69 42 46 94	73 90 1 0 96 69 99	44 22 92 28 42 15	43 80 97 1 42 14 13	21 7 68 82 11 94 23		

33	49	12	1	92	97	68
48	78	69	0	28	1	82
94	91	42	96	42	42	11
4	83	46	69	15	14	94
64	34	94	99	1	13	23
21811	29446	26742	30700	12807	15432	23296
19384	25094	18845	22196	20744	23210	24071
21609	31522	25566	27468	21792	26862	28351
20497	26153	22205	33136	11855	16842	16802
21215	20226	20969	29476	17000	25625	14813
23628	29093	28312	27777	18328	22863	21659
19248	21626	22783	22424	11113	16427	15371
16728	20530	19673	15144	18050	19444	21028
21811	29446	26742	30700	12807	15432	23296
19384	25094	18845	22196	20744	23210	24071
21609	31522	25566	27468	21792	26862	28351
20497	26153	22205	33136	11855	16842	16802
21215	20226	20969	29476	17000	25625	14813
23628	29093	28312	27777	18328	22863	21659
19248	21626	22783	22424	11113	16427	15371
16728	20530	19673	15144	18050	19444	21028
	48 94 4 64 21811 19384 21609 20497 21215 23628 19248 21811 19384 21609 20497 21215 23628 19248	48 78 94 91 4 83 64 34 21811 29446 19384 25094 21609 31522 20497 26153 21215 20226 23628 29093 19248 21626 16728 20530 21811 29446 19384 25094 21609 31522 20497 26153 21215 20226 23628 29093 19248 21626	48 78 69 94 91 42 4 83 46 64 34 94 21811 29446 26742 19384 25094 18845 21609 31522 25566 20497 26153 22205 21215 20226 20969 23628 29093 28312 19248 21626 22783 16728 20530 19673 21811 29446 26742 19384 25094 18845 21609 31522 25566 20497 26153 22205 21215 20226 20969 23628 29093 28312 19248 21626 20969 23628 29093 28312 19248 21626 22783	48 78 69 0 94 91 42 96 4 83 46 69 64 34 94 99 21811 29446 26742 30700 19384 25094 18845 22196 21609 31522 25566 27468 20497 26153 22205 33136 21215 20226 20969 29476 23628 29093 28312 27777 19248 21626 22783 22424 16728 20530 19673 15144 21811 29446 26742 30700 19384 25094 18845 22196 21609 31522 25566 27468 20497 26153 22205 33136 21215 20226 20969 29476 23628 29093 28312 27777 19248 21626 22783 22424	48 78 69 0 28 94 91 42 96 42 4 83 46 69 15 64 34 94 99 1 21811 29446 26742 30700 12807 19384 25094 18845 22196 20744 21609 31522 25566 27468 21792 20497 26153 22205 33136 11855 21215 20226 20969 29476 17000 23628 29093 28312 27777 18328 19248 21626 22783 22424 11113 16728 20530 19673 15144 18050 21811 29446 26742 30700 12807 19384 25094 18845 22196 20744 21609 31522 25566 27468 21792 20497 26153 22205 33136 11855 21215 20226 20969 29476 17000	48 78 69 0 28 1 94 91 42 96 42 42 4 83 46 69 15 14 64 34 94 99 1 13 21811 29446 26742 30700 12807 15432 19384 25094 18845 22196 20744 23210 21609 31522 25566 27468 21792 26862 20497 26153 22205 33136 11855 16842 21215 20226 20969 29476 17000 25625 23628 29093 28312 27777 18328 22863 19248 21626 22783 22424 11113 16427 16728 20530 19673 15144 18050 19444 21811 29446 26742 30700 12807 15432 19384 25094 18845 22196 20744 23210 21609 31522 25566 <t< td=""></t<>

size = 16x16

[CPU]

Clock Rate (MHz): 9.000 Clock Count (tick): 28 Time (second): 0.000003

[GPU]

Clock Rate (MHz): 9.000
Clock Count (tick): 707
Time (second): 0.000071

size = 32x32

[CPU]

Clock Rate (MHz): 9.000 Clock Count (tick): 155 Time (second): 0.000016

[GPU]

Clock Rate (MHz): 9.000
Clock Count (tick): 706
Time (second): 0.000071

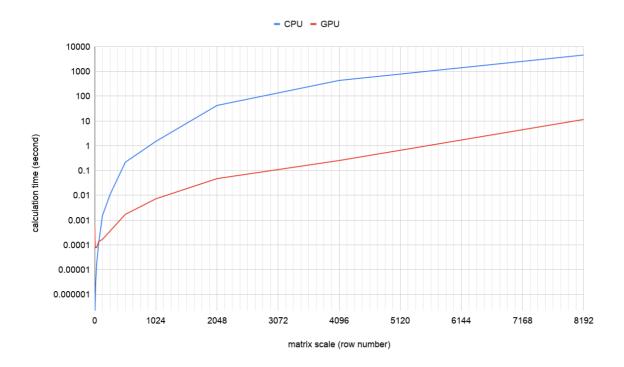
size = 64x64

```
[CPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 983
Time (second): 0.000098
[GPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 1228
Time (second): 0.000123
_____
size = 128x128
[CPU]
clock Rate (MHz): 9.000
Clock Count (tick): 13668
Time (second): 0.001367
[GPU]
clock Rate (MHz): 9.000
Clock Count (tick): 1500
Time (second): 0.000150
size = 256x256
[CPU]
clock Rate (MHz): 9.000
Clock Count (tick): 96716
Time (second): 0.009672
[GPU]
clock Rate (MHz): 9.000
Clock Count (tick): 3256
Time (second): 0.000326
size = 512x512
[CPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 1954040
Time (second): 0.195404
[GPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 15381
Time (second): 0.001538
_____
size = 1024x1024
[CPU]
clock Rate (MHz): 9.000
```

```
Clock Count (tick): 13599900
Time (second): 1.359990
[GPU]
clock Rate (MHz): 9.000
Clock Count (tick): 65413
Time (second): 0.006541
size = 2048x2048
[CPU]
clock Rate (MHz): 9.000
Clock Count (tick): 381469500
Time (second): 38.146950
[GPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 426655
Time (second): 0.042666
size = 4096x4096
[CPU]
clock Rate (MHz): 9.000
Clock Count (tick): 3906021319
Time (second): 390.602132
[GPU]
clock Rate (MHz): 9.000
Clock Count (tick): 2290312
Time (second): 0.229031
_____
size = 8192x8192
[CPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 41306953655
Time (second): 4130.695365
[GPU]
Clock Rate (MHz): 9.000
Clock Count (tick): 103745836
Time (second): 10.374584
```

5. Conclusion

The performance of matrix dot production is shown as the following chart. **Notice that the vertical axis is log scale.**



The chart shows that:

- When the matrix size is larger than 128x128, GPU spends less time than CPU.
- When the matrix size is larger than 1024x1024, the time spending of CPU is 2.5-3 orders of magnitude than which of GPU.